Recursion

Recursion

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- Recursion:
  - When a method/function calls itself
- A recursive method is a method that calls itself

Recursion

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- We are defining a method named **add**

Recursion

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

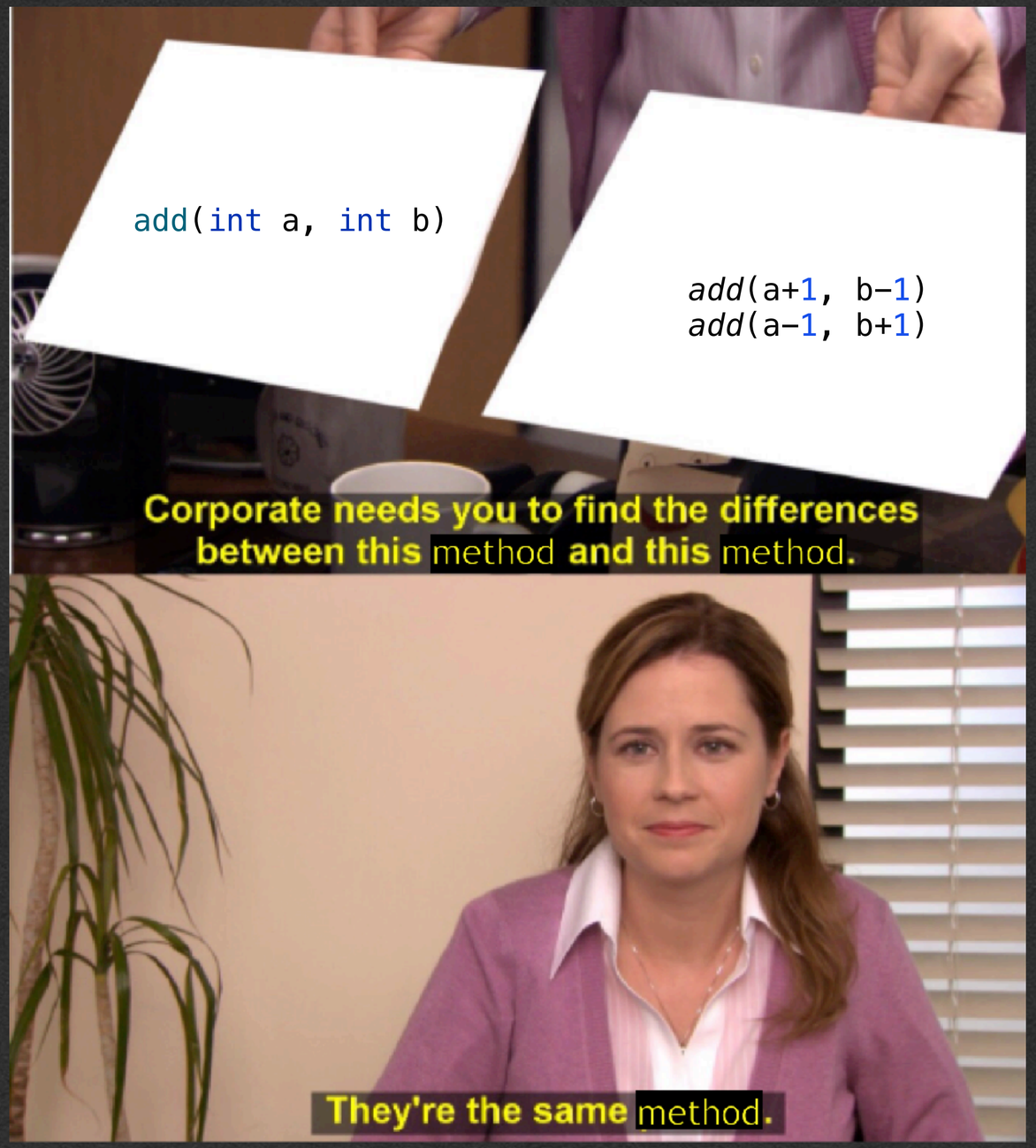
- We are defining a method named **add**
- And we call a method named **add**

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



Recursion

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- This **add** method calls itself!
- This is a recursive method.

Recursion

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- Let's see how this work with a...

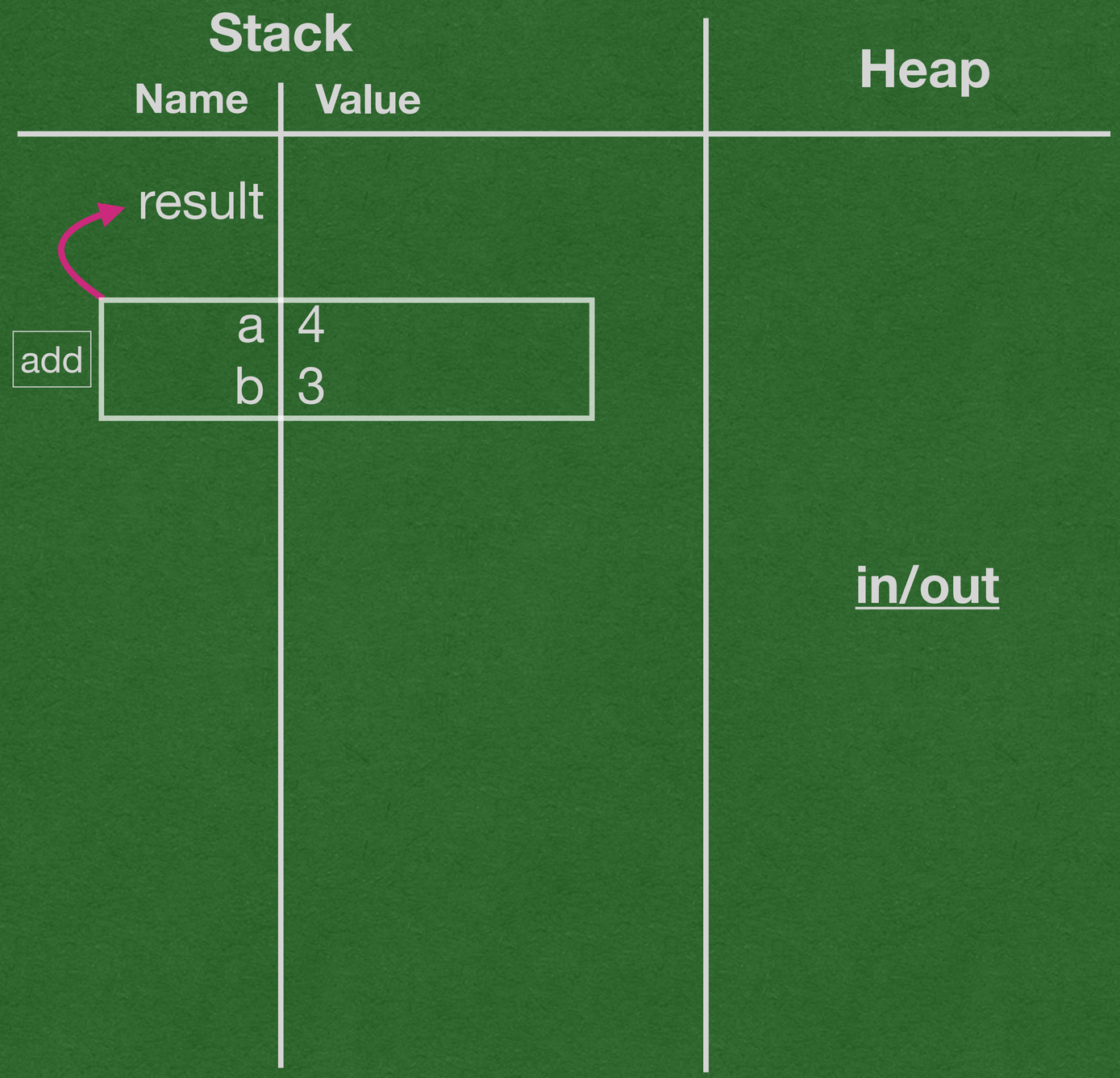
Memory Diagram!

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



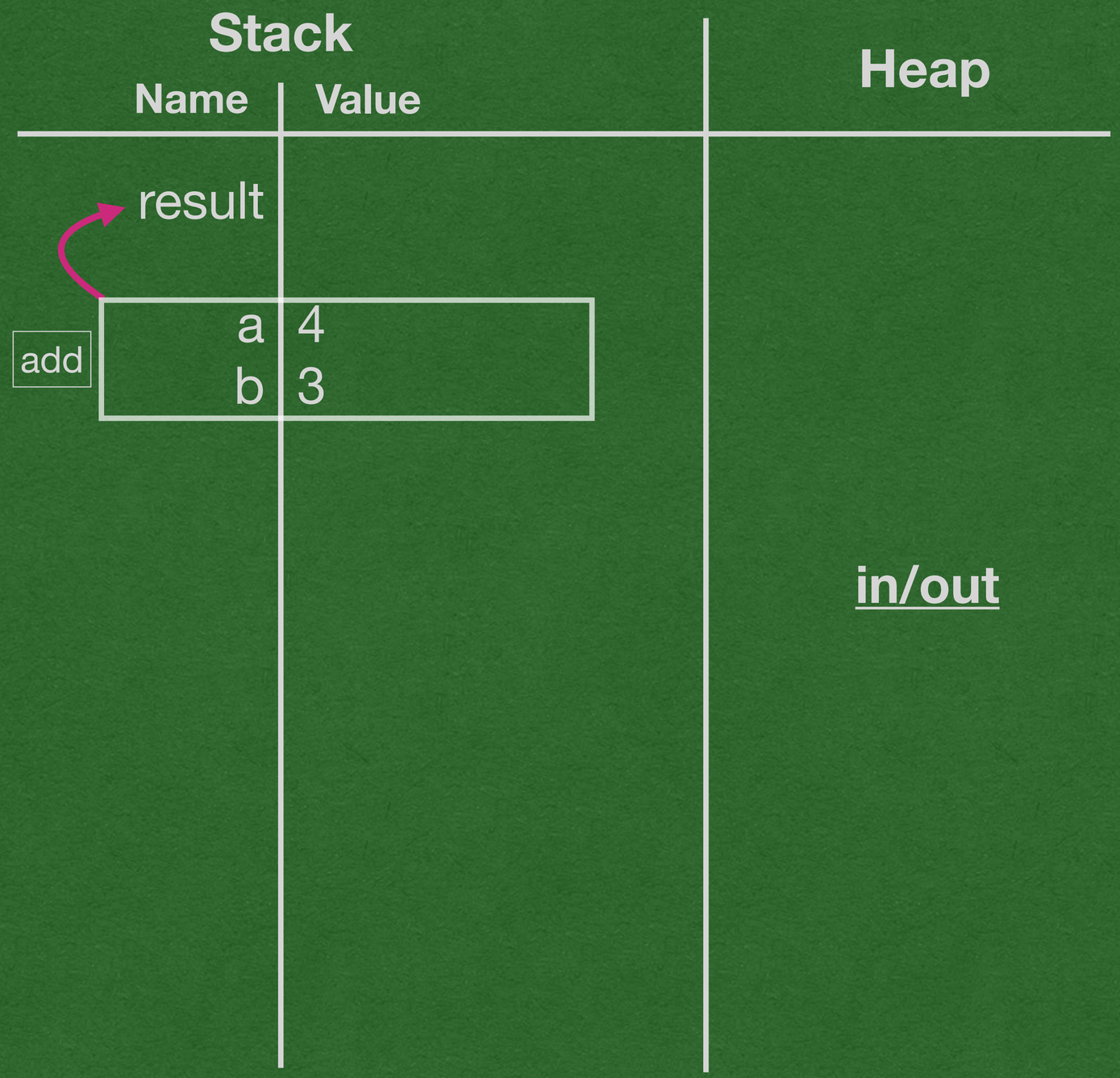
- The diagram starts with a method call
- Add a stack frame on the stack with the parameters of the method call

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



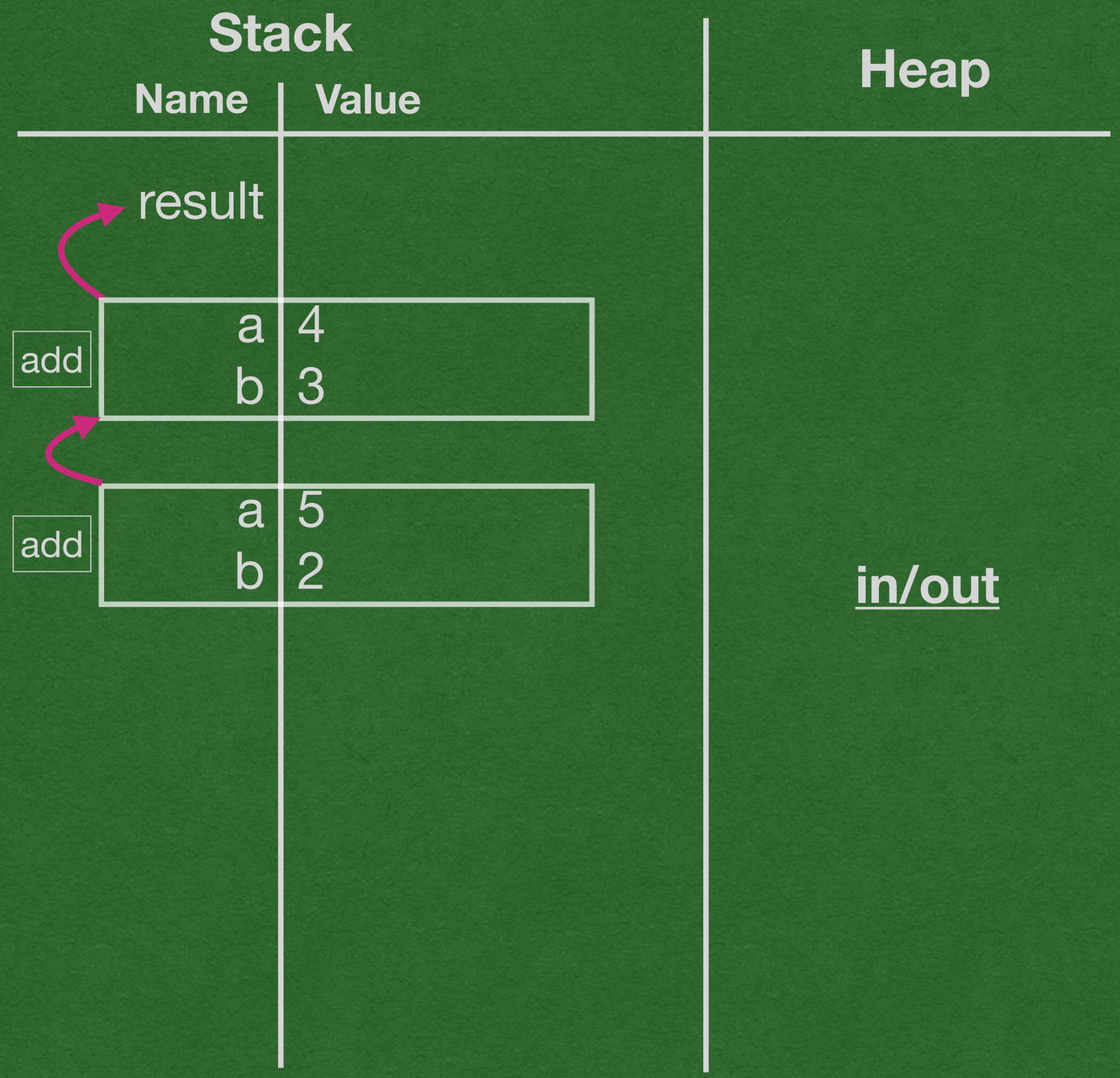
- We reach the recursive call
- The trick:
  - There is none. Treat this as any other method call

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



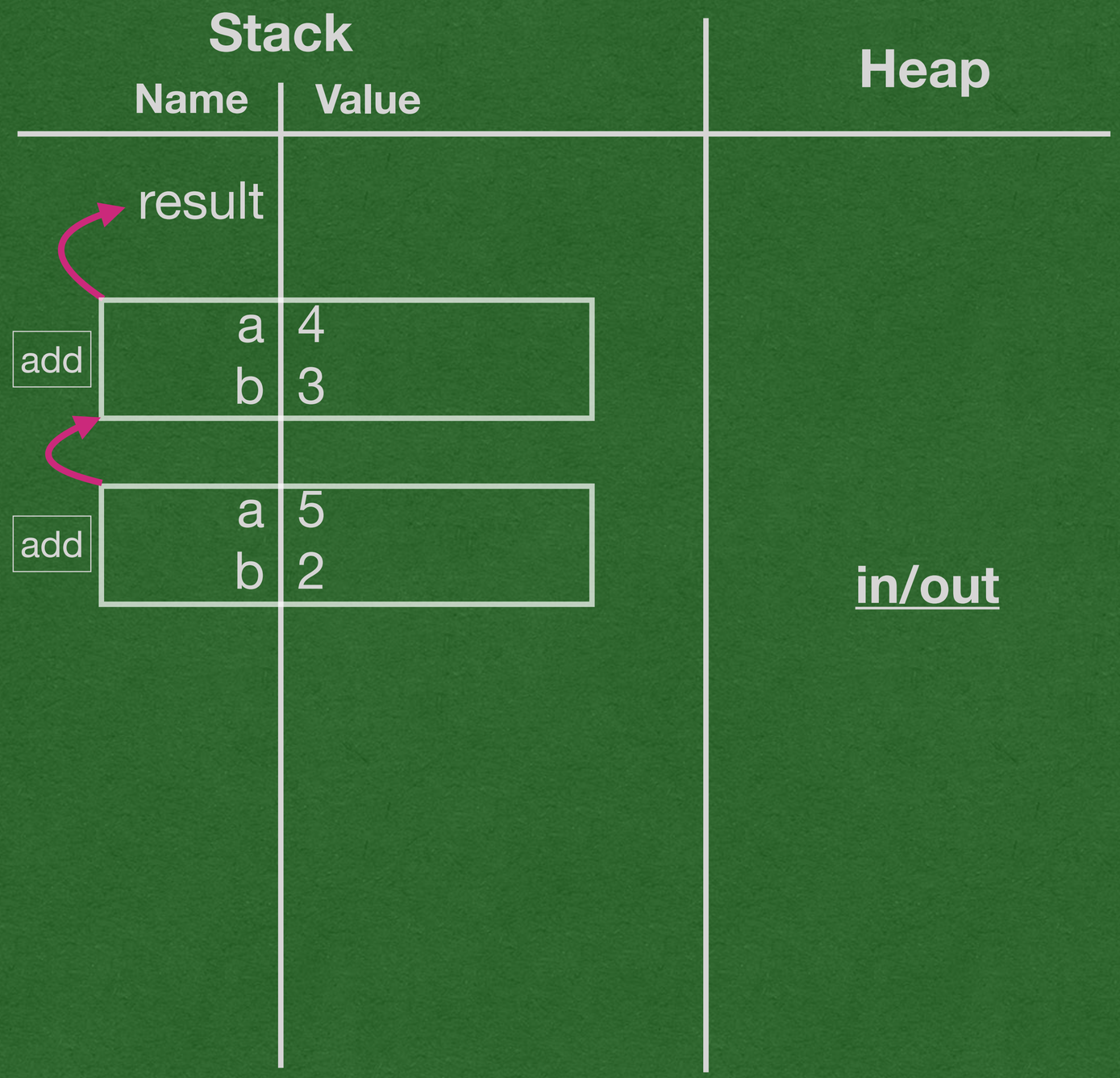
- Add the stack frame a parameters to the stack just like any other method call
- The return arrow points to the stack frame that called the method

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



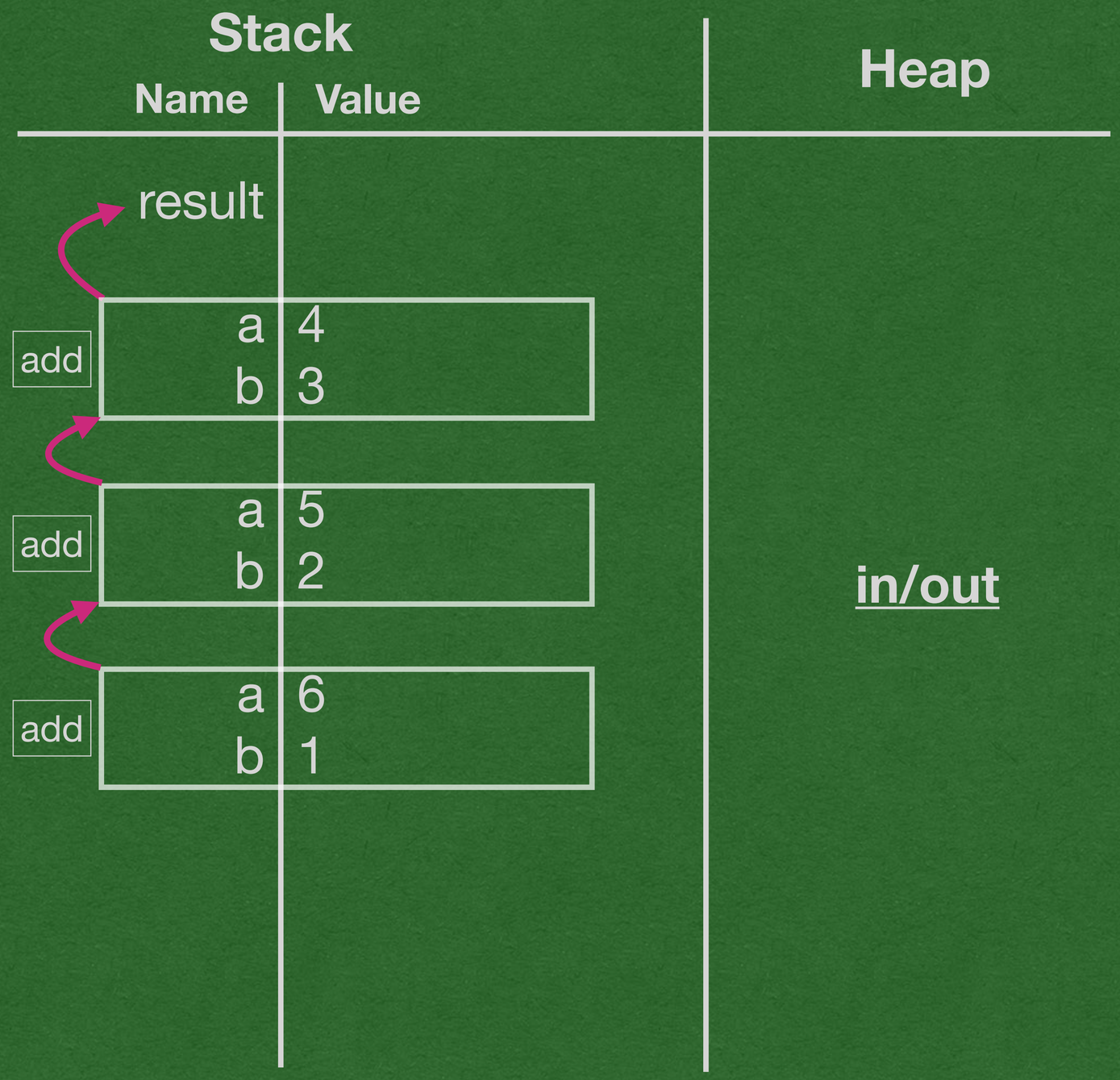
- We get to the next recursive call
- Do it again!

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



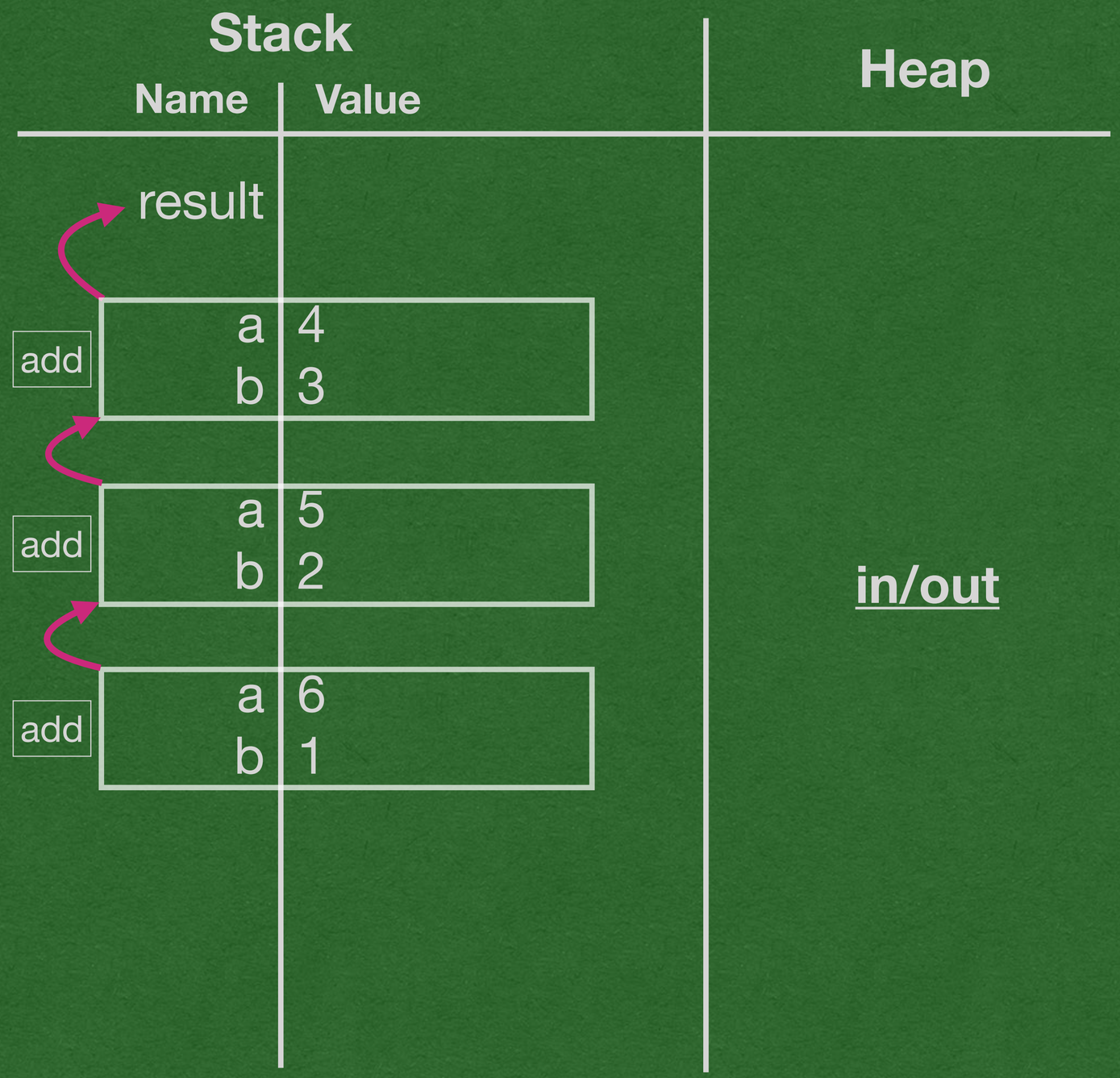
- We have 3 frames on the stack (plus the main stack frame)
- Only the frame on the top of the stack is active

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



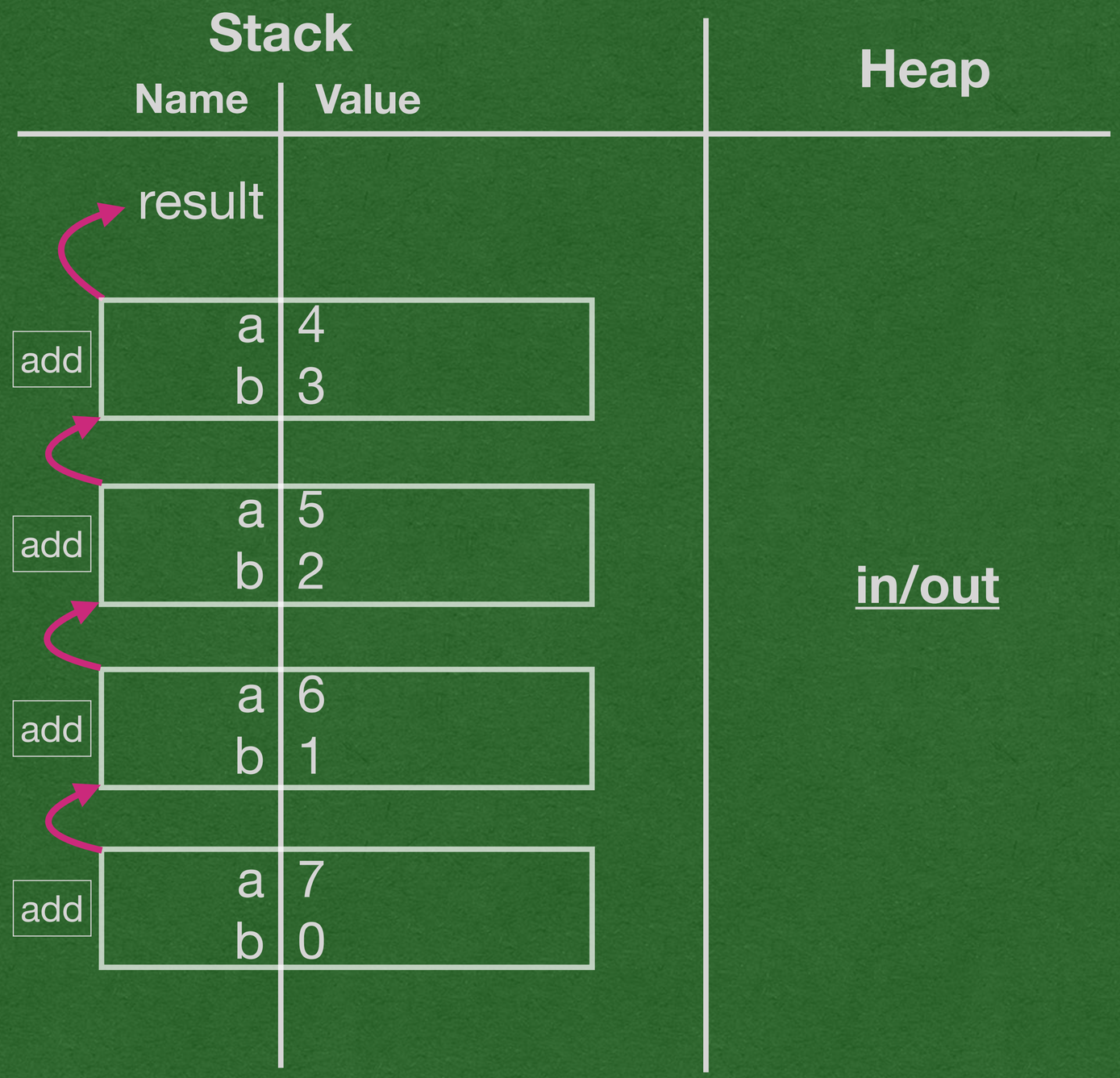
- The other stack frames are waiting until they are back on top of the stack

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



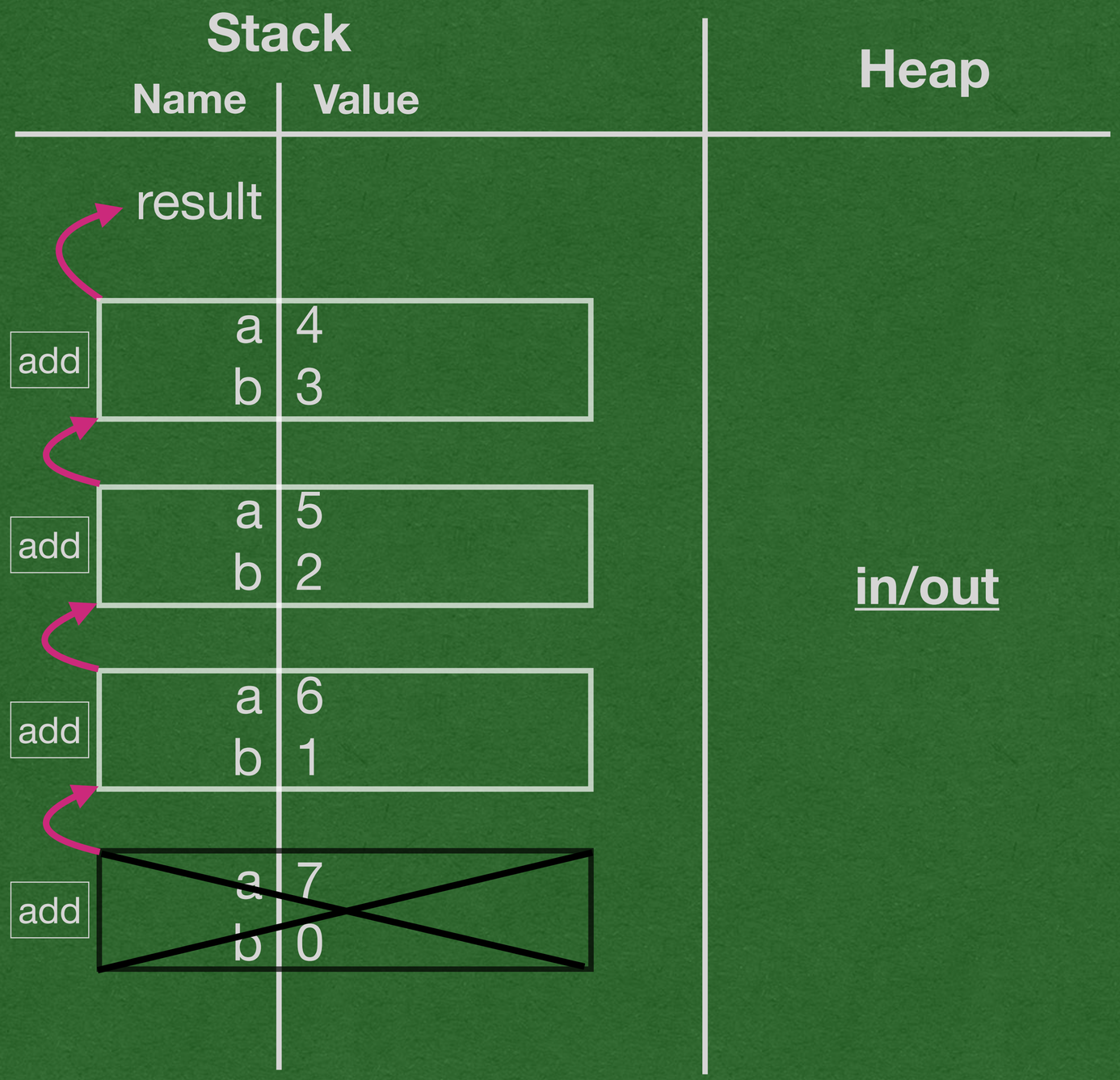
- Call the method again
- This time, the first condition is true

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



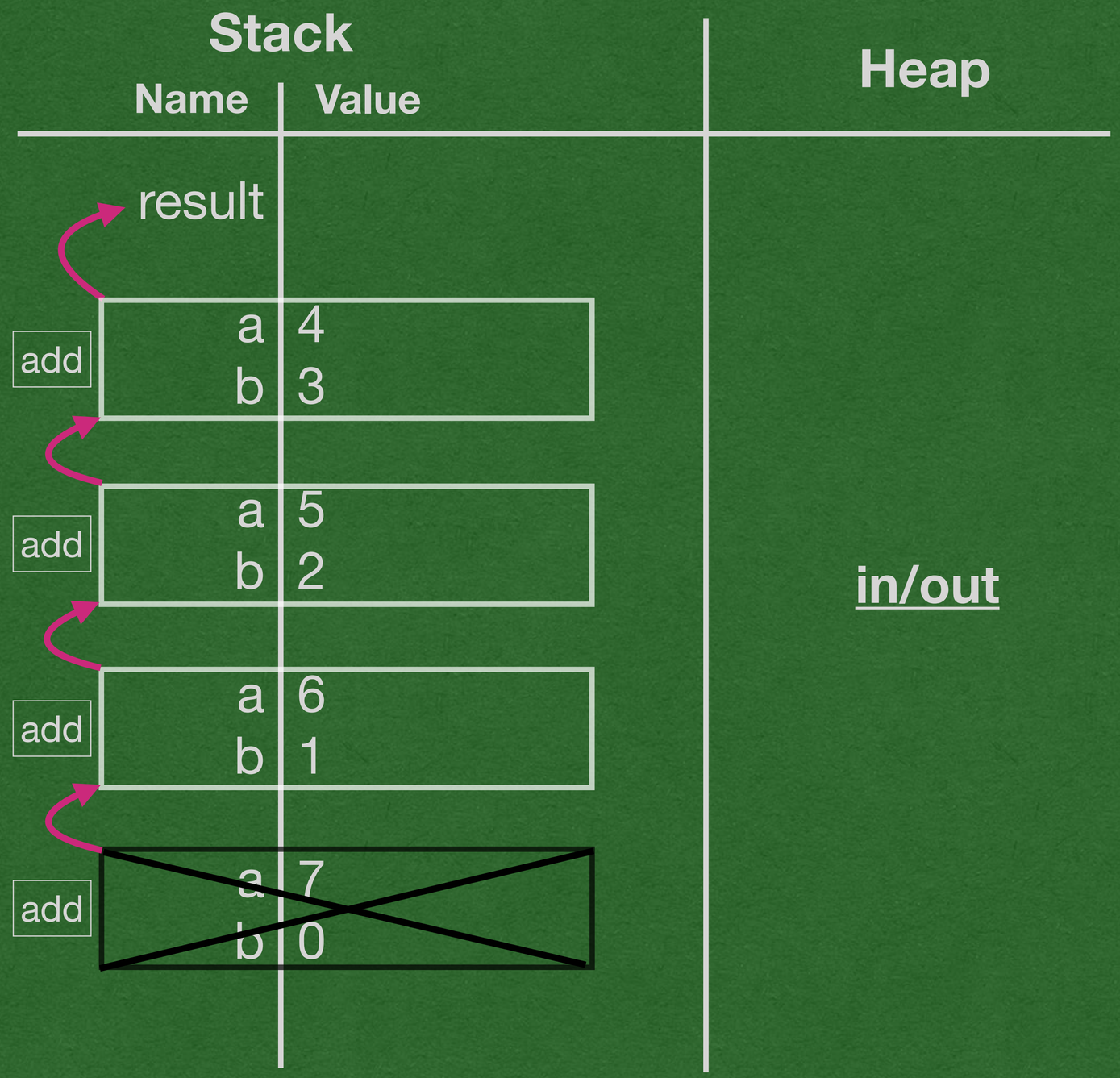
- This stack frame returns 7
- The frame is removed from the stack and the next frame regains control

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



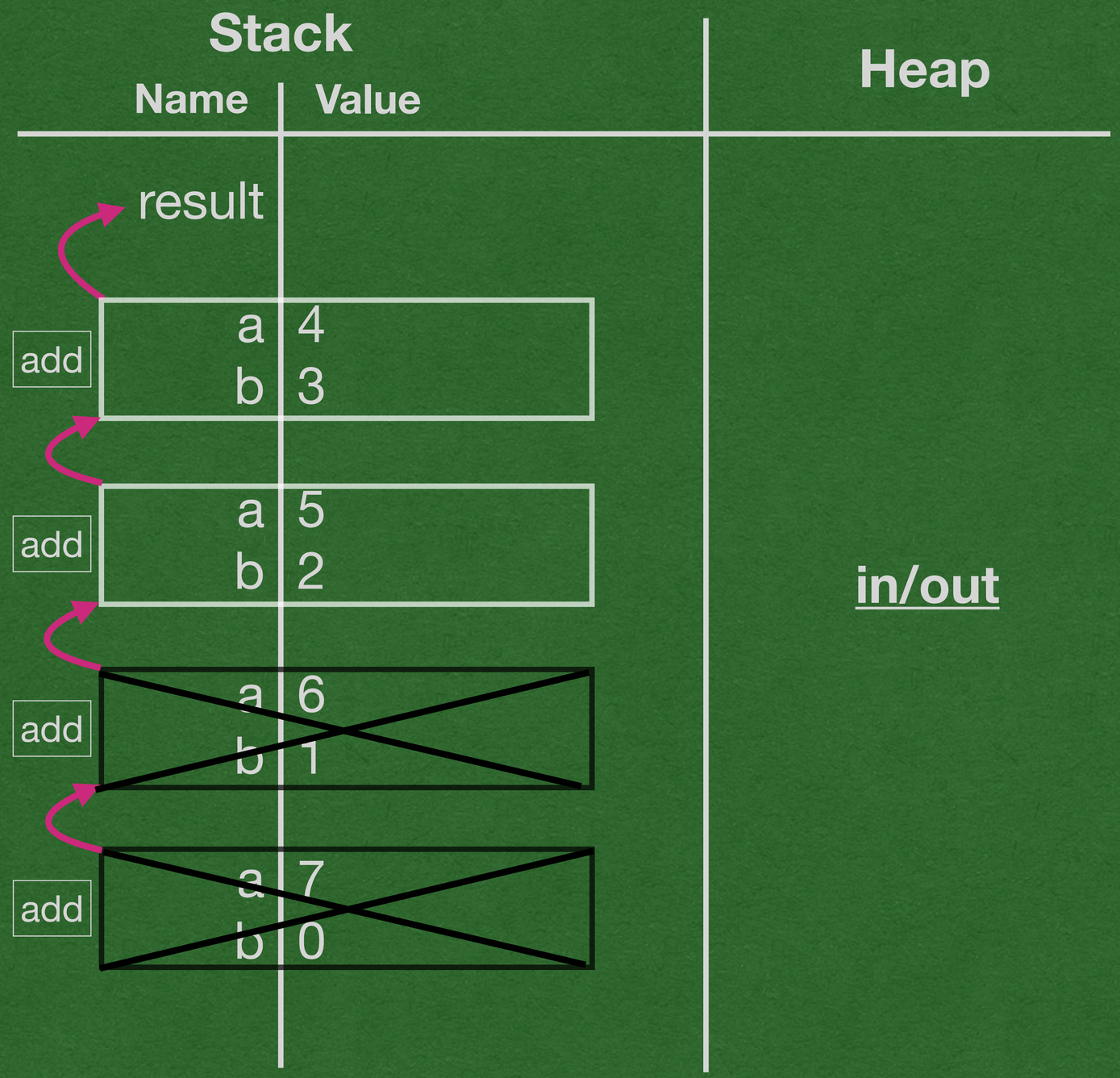
- This frame called add(6, 1)
  - The method call evaluated to 7

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



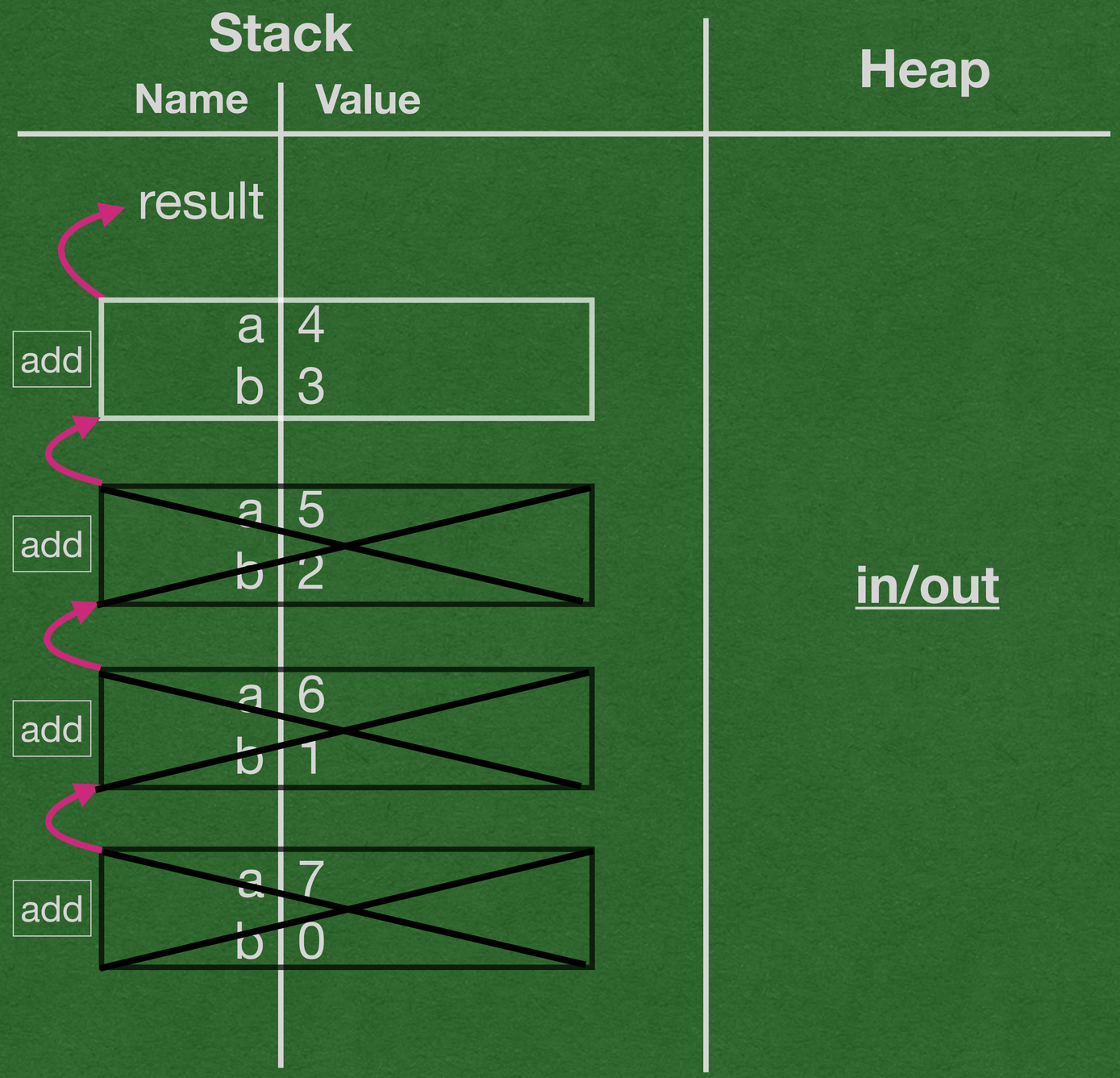
- Returns the value 7
- This frame is removed from the stack
- Control goes to the next frame on the stack

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



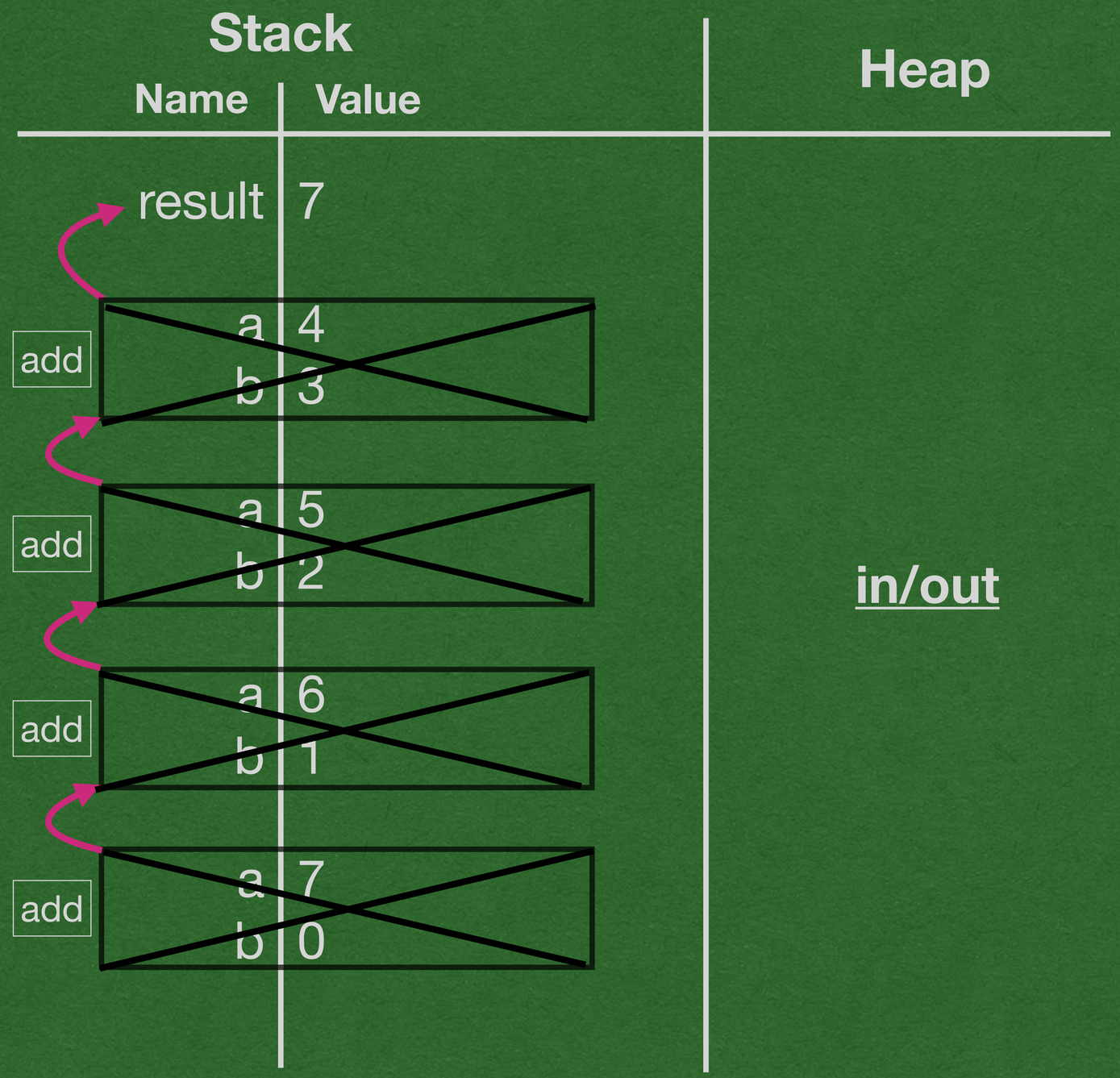
- The next frame returns the value 7 that it received from the previous frame

```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```



- Return 7 to the main stack frame

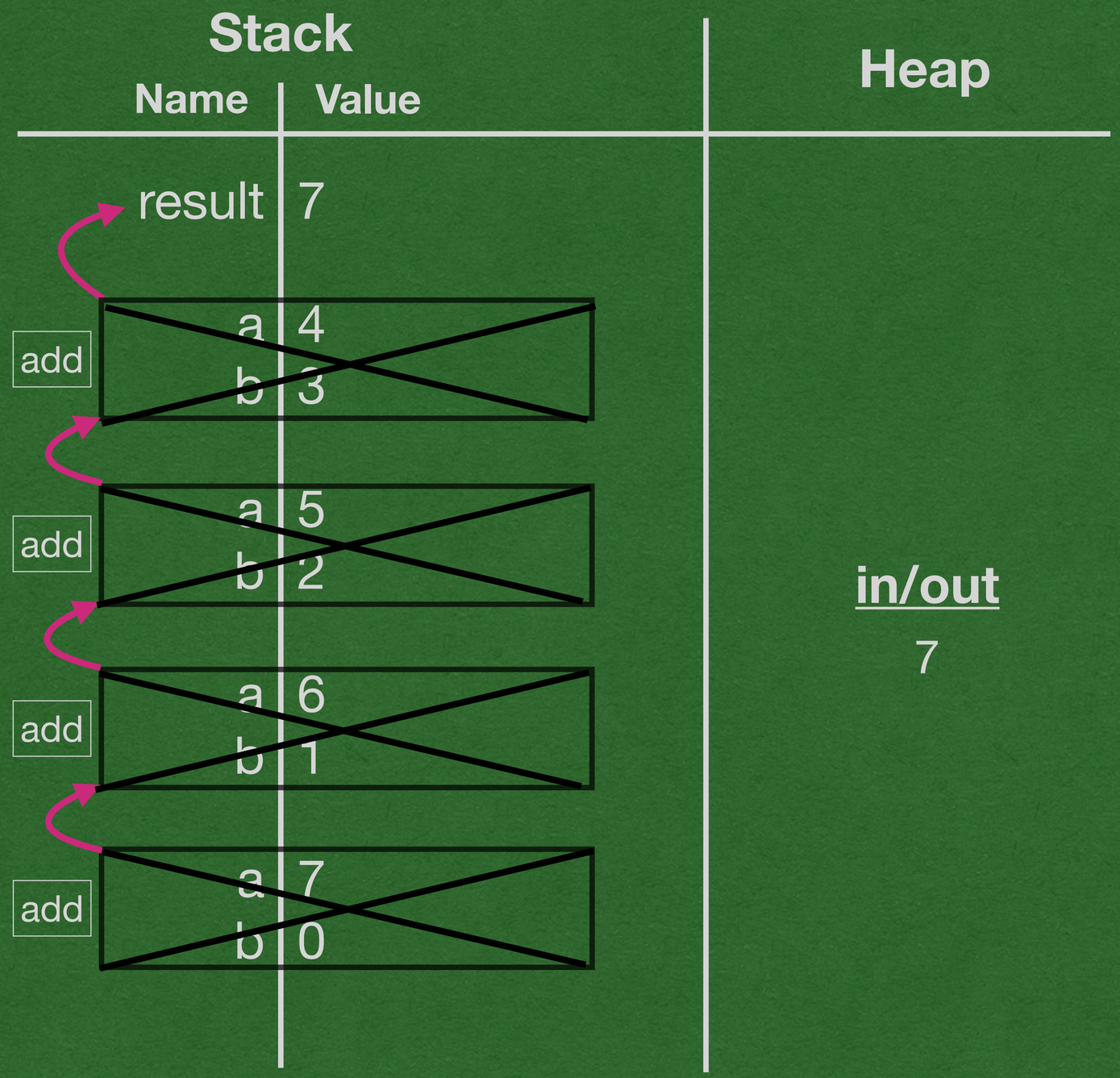
```java
package week2;

public class FirstRecursion {

    public static int add(int a, int b) {
        if (b == 0) {
            return a;
        } else if (b > 0) {
            return add(a+1, b-1);
        } else {
            return add(a-1, b+1);
        }
    }

    public static void main(String[] args) {
        int result = add(4, 3);
        System.out.println(result);
    }
}
```

- Print 7 to the screen
- End the program



Interpretation
vs
Compilation

Interpretation vs. Compilation

- Interpretation
  - Code is read and executed one statement at a time
- Compilation
  - Entire program is translated into another language
  - The translated code is interpreted

Interpretation

- Python, JavaScript, etc. are interpreted languages

- If you have errors:
  - They'll commonly be *run-time* errors
  - Program crashes as it's running

- Program runs immediately when you run it

Compilation

- Java, C, Scala, C++, etc. are compiled languages

- If you have errors:
  - They'll commonly be *compiler* errors
  - Compilers will check all syntax and types and alert us of any errors before they become run-time errors
  - Program fails to be converted into the target language and never runs

- Compilation takes time; Program does not run immediately

Compilation - Java

- Java compiles to Java Byte Code
- Executed by the Java Virtual Machine (JVM)
  - Installed on Billions of devices!